(12) DEMANDE INTERNATIONALE PUBLIÉE EN VERTU DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)

(19) Organisation Mondiale de la Propriété Intellectuelle
Bureau international

(43) Date de la publication internationale
25 mai 2001 (25.05.2001)

PCT

(10) Numéro de publication internationale
WO 01/37085 A1

(51) Classification internationale des brevets[7]: G06F 9/445, G07F 7/10

(21) Numéro de la demande internationale:
PCT/FR00/03193

(22) Date de dépôt international:
17 novembre 2000 (17.11.2000)

(25) Langue de dépôt: français

(26) Langue de publication: français

(30) Données relatives à la priorité:
99/14454 17 novembre 1999 (17.11.1999) FR

(71) Déposant *(pour tous les États désignés sauf US)*: **BULL CP8** [FR/FR]; 68, route de Versailles, Boîte postale 45, F-78430 Louveciennes (FR).

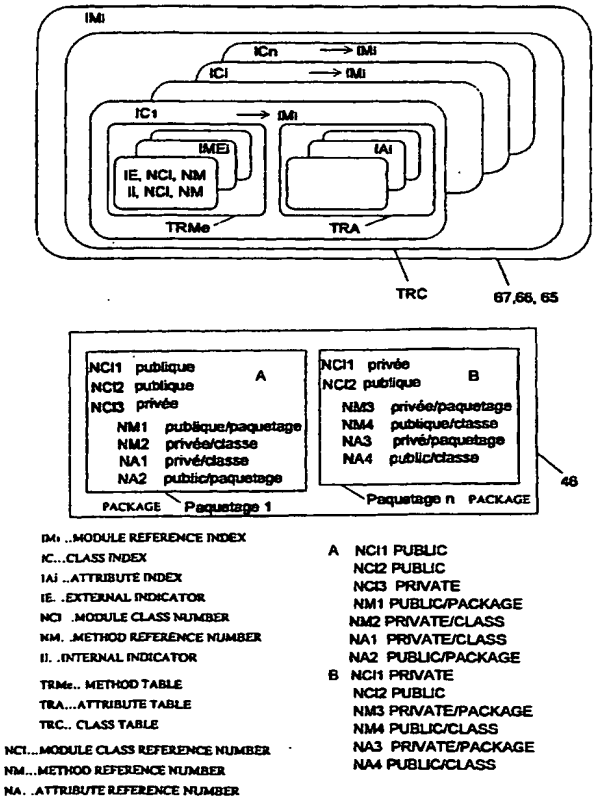
(72) Inventeurs; et

(75) Inventeurs/Déposants *(pour US seulement)*: **GOIRE, Christian** [FR/FR]; 8, allée du Mail, F-78340 Les Clayes sous Bois (FR). **BILLON, Jean-Paul** [FR/US]; 96, Uranus terrace, San Francisco, CA 94114 (US).

[Suite sur la page suivante]

(54) Title: METHOD FOR LOADING APPLICATIONS IN A MULTIAPPLICATION ONPLATFORM SYSTEM EQUIPPED WITH DATA PROCESSING RESOURCES, CORRESPONDING EXECUTING SYSTEM AND METHOD

(54) Titre: PROCEDE DE CHARGEMENT D'APPLICATIONS DANS UN SYSTEME EMBARQUE MULTI-APPLICATION MUNI DE RESSOURCES DE TRAITEMENT DE DONNEES, SYSTEME, ET PROCEDE D'EXECUTION CORRESPONDANTS



(57) Abstract: The invention concerns a method for loading application in a multiapplication onplatform system, the corresponding onplatform system, and the method for executing an application of the onplatform system. The system for application loading on an onplatform system comprising a execution environment including a virtual machine comprising a language interpreter of the intermediate pseudocode type, application programming interfaces (API), from a station whereon the application source code is written, compiled in pseudocode, verified and converted is characterised in that the conversion comprises steps which consist in: producing a static linkage of a plurality of packages to be stored in the same name space on the onplatform system, called modules, and constituting an application programming interface module or a service module corresponding to an application, and consists in assigning an identifier to each module (MID), and a reference number to each class (NCI), to each method (NM) and to each attribute (NA). The reference to a method or an attribute in the pseudocode linked to a module is coded on three multiplets formed by an indicator signalling the reference to a class internal (II) or an external (IE) to the module, the class number (NCI) and either the method number (NM) or the attribute number (NA), a reference (IE) to an external class being systematically interpreted by the virtual machine as a reference to an application programming interface module.

[Suite sur la page suivante]

WO 01/37085 A1

(74) **Mandataire:** **CORLU, Bernard**; Bull S.A., PC58D20, 68, route de Versailles, F-78434 Louveciennes Cedex (FR).

(81) **États désignés** *(national)*: BR, CA, CN, JP, US.

(84) **États désignés** *(régional)*: brevet européen (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR).

Publiée:
— *Avec rapport de recherche internationale.*

En ce qui concerne les codes à deux lettres et autres abréviations, se référer aux "Notes explicatives relatives aux codes et abréviations" figurant au début de chaque numéro ordinaire de la Gazette du PCT.

(57) **Abrégé:** La présente invention concerne un procédé de chargement d'applications dans un système embarqué multi-application, le système embarqué correspondant, et le procédé d'exécution d'une application du système embarqué. Le procédé de chargement d'applications sur un système embarqué comprenant un environnement d'exécution incluant une machine virtuelle comprenant un interpréteur de langage de type pseudocode intermédiaire, des interfaces de programmation d'application (API), à partir d'une station sur laquelle le code source de l'application est écrit, compilé en pseudocode, vérifié et converti, se caractérise n ce que la conversi n comprend la réalisati n de la liaison statique d'une pluralité d'ensemble de paquetages destinés à être stockés dans le même espace nom sur le système embarqué, appelés modules, et constituant un module d'interface de programmation d'application ou un module de service correspondant à une application, et consiste à attribuer un identificateur à chaque module (MID), et un numéro de référence à chaque classe (NCI), à chaque méthode (NM) et à chaque attribut (NA). La référence à une méthode ou un attribut, dans le pseudocode lié d'un module est codée sur trois multiplets constitués par un indicateur indiquant la référence à une classe interne (II) ou externe (IE) au module, le numéro de la classe (NCI) et soit le numéro de la méthode (NM) soit le numéro de l'attribut (NA), une référence (IE) à une classe externe étant systématiquement interprétée par la machine virtuelle comme une référence à un module d'interface de programmation d'application.

PROCEDE DE CHARGEMENT D'APPLICATIONS DANS UN SYSTEME EMBARQUE MULTI-APPLICATION MUNI DE RESSOURCES DE TRAITEMENT DE DONNEES, SYSTEME, ET PROCEDE D'EXECUTION CORRESPONDANTS

La présente invention concerne un procédé de chargement d'applications dans un système embarqué multi-application muni de ressources de traitement de données, le système embarqué correspondant, et le procédé d'exécution d'une application d'un système embarqué correspondant.

La présente invention concerne plus particulière la réalisation de pare-feu entre modules partageant le même espace mémoire dans des systèmes embarqués sur des objets portables multi-application utilisant un pseudocode intermédiaire et une machine virtuelle associée.

La technologie "Java" (marque déposée) introduite par la société "Sun" est basée sur un langage de programmation orienté objet "Java" et une machine virtuelle associée. Cette technologie a été développée sur des stations ou PC (Personal Computer), appelées ci-après plate-forme "Java" classique, possédant une puissance CPU et une mémoire importante de l'ordre du méga byte ou mégaoctet.

Depuis quelques années, les concepts de la technologie "Java" ont été repris et adaptés pour fonctionner sur des systèmes embarqués dans des objets portables, par exemple au format de carte de crédit ou micromodule SIM incorporant un microprocesseur et appelée communément carte à puce, ou encore de téléphones portables GSM (Global System for Mobile Communication), cartes PCMCIA ou tous autres terminaux portables. Par la suite nous utiliserons le terme carte pour désigner l'un quelconque de ces objets portables. La programmation des systèmes embarqués, jusqu'ici réalisée en assembleur, est désormais possible dans un langage évolué comme "Java", et permet de faciliter et d'accélérer le développement d'applications clientes.

Ce nouveau type de plate-forme spécifique aux systèmes embarqués d'un objet portable, appelé ci-après plate-forme spécifique, constitue un sous-ensemble de la plate-forme classique. Les différences résultent du fait de l'environnement réduit des systèmes embarqués. De manière classique, tel que représenté à la figure 4a, une carte à puce (10) comprend un système d'entrée et de sortie (11) relié au microprocesseur (14), une mémoire volatile RAM (12) (Random Access Memory) une mémoire non volatile constituée par une mémoire morte ROM (13) (Read Only Memory) et une mémoire non volatile programmable (15) constituée d'une flash RAM ou EEPROM (Electrically Erasable Programmable Read Only Memory). L'ensemble de ces éléments est relié au microprocesseur par un BUS de liaison. A titre d'exemple, une carte à puce comprend dans le meilleurs des cas, en utilisant les nouveaux composants actuellement existants, une mémoire ROM, une mémoire EEPROM de 32 kilooctets ou kilo bytes, et une mémoire RAM de 2 Kilo Bytes.

Dans un système "Java" classique, une application est écrite sur une station ou PC. Son code source est compilé dans un pseudocode intermédiaire, appelé "Java Byte Code" ou byte code, qui est indépendant du code machine de la plate-forme utilisée. L'application est ensuite téléchargée sur la plate-forme cible ou plate-forme "Java" classique. L'application chargée, constituée d'un ensemble de classes dites classes clients dont les méthodes ont été compilées dans le pseudocode, s'appuie sur les interfaces de programmation applicatives ou Interface pour la Programmation d'Application ou API (Application Programming Interface). Les interfaces de programmation applicatives permettent d'uniformiser l'interface utilisateur, de contrôler un éditeur, un clavier ou une imprimante par exemple. La machine virtuelle d'une plate-forme classique comprend un vérifieur de pseudocode, un chargeur dynamique de classe, un interpréteur de pseudocode qui permet la traduction en code machine et un gestionnaire de sécurité.

La principale différence entre une machine virtuelle classique et une machine virtuelle d'un système embarqué, appelée machine virtuelle

spécifique, est due au fait que la machine virtuelle d'un système embarqué est décomposée en deux parties séparées. Suivant la figure 4b, la machine virtuelle comprend une partie (30) hors de la plate-forme spécifique (40), appelée machine virtuelle hors plate-forme ("off-platform"), comprenant un convertisseur (32), et une partie (41) dans la carte constituant la plate-forme spécifique (40), appelée machine virtuelle embarquée (41) ("on-platform") incluant l'interpréteur de pseudocode.

Ainsi, dans le cas d'une plate-forme spécifique, le programme source (21) de l'application est écrit, compilé en pseudocode intermédiaire par un compilateur (22), et vérifié par un vérifieur (31) de pseudocode intermédiaire sur une station (20) classique, puis converti par le convertisseur (32), placé sur la même station (20) ou une autre station. Après un éventuel passage par un signeur (34), l'application est ensuite téléchargée sur la mémoire volatile programmable électriquement et éventuellement effaçable électriquement EEPROM (15) de l'objet portable ou plate-forme spécifique (40). Ce chargement est effectué par un chargeur comportant une partie hors plate-forme appelée téléchargeur (33) et une partie sur la plate-forme spécifique appelée chargeur (42). Contrairement à ce qui existe sur une plate-forme classique pour une station, la machine virtuelle (41) d'une plate-forme spécifique (40), placée en mémoire ROM avec le système d'exploitation (48) (Operating System) ne peut comporter de vérifieur de pseudocode intermédiaire, celui-ci étant trop lourd pour être stocké et/ou exécuté dans l'objet portable. La plate-forme spécifique (40) ne contient pas non plus de chargeur dynamique de classes. En effet, pour le domaine d'application de l'invention, sur la machine virtuelle (30) hors plate-forme, le vérifieur (31) vérifie que les classes compilées sont bien formées et vérifie les violations de langage spécifiques à la description de la plate-forme spécifique. Le convertisseur (32) effectue le travail requis pour le chargement des classes et la résolution des références. Le convertisseur effectue la liaison statique des classes, il résout les références symboliques aux classes, méthodes et attributs déjà présents sur la carte. Il répartit le

stockage et crée les structures des données pour représenter les classes, crée les méthodes et attributs statiques ou natifs et initialise les variables statiques.

L'environnement d'exécution de la plate-forme spécifique (Runtime Environment ou RE) comprend la machine virtuelle embarquée ("on-platform") (41) se limitant à un interpréteur, une plate-forme API et les méthodes dites natives (43) ou encore statiques associées. La plate-forme API comprend les API (44) (Application Programming Interface) définissant un ensemble de classes, dites classes système (45), et les conventions d'appel par lesquelles une application accède à l'environnement d'exécution (RE) et aux méthodes statiques (43). Les méthodes statiques (43) exécutent les services d'allocation de mémoire, d'entrée et sortie, et cryptographique de la carte.

L'interpréteur de la machine virtuelle embarquée de la carte (41)(on-platform), sert de support au langage "Java", et lit de façon séquentielle le pseudocode intermédiaire, instruction par instruction. Chaque instruction standard de ce pseudocode intermédiaire est interprétée dans le langage du microprocesseur par l'interpréteur puis exécutée par le microprocesseur. En règle générale, les instructions standards du pseudocode intermédiaire permettent de traiter des fonctions évoluées telle que le traitement arithmétique et la manipulations d'objets. La notion d'objet concerne les objets informatiques tels que listes, tableaux de données ou analogues. Les classes dites clients (46) des applications et les classes systèmes (45) des API sont toutes chargées dans le même espace mémoire et sont gérées par l'intermédiaire d'une table de classe (47).

La machine virtuelle (41) est également chargée de gérer les classes et objets et de faire respecter les séparations ou pare-feu entre les applications afin de permettre un partage sécurisé des données, appelées également attributs, variables ou champs (fields).

Dans le cas d'un objet portable de type carte à puce, une application d'une plate-forme spécifique peut être activée directement par

l'environnement d'exécution (RE) lorsqu'une APDU (Application Protocol Data Unit) de sélection émise par un service ou terminal est reçue par la carte.

Afin de restreindre l'accès aux données entre les parties de code partageant le même espace mémoire, une des méthodes classiques sur une plate-forme classique est basée sur la restriction explicite de visibilité déclarée dans le code source. Suivant la figure 4c, les méthodes (NM1, NM2), respectivement (NM3, NM4), et les attributs (NA1, NA2), respectivement (NA3, NA4) sont encapsulés dans des classes (NCl3), respectivement (NCl2), qui elles-mêmes font partie de paquetages (Paquetage 1), respectivement (Paquetage n) regroupant chacun plusieurs classes. Une classe peut être publique telle que (NCl1 ou NCl2) ou privée telle que (NCl3) par rapport à un paquetage, ce qui implique, dans le dernier cas, que seules les classes du même paquetage peuvent accéder à cette classe. Les méthodes (exemple NM2) et les données (exemple NA1) d'une classe (NCl3) peuvent être privées par rapport à la classe (NM2, NA1), qui elle-même est privée par rapport au paquetage (Paquetage 1), ou publiques par rapport au paquetage (par exemple NM1, NA2) ou à la classe (par exemple NM4, NA4). Cette restriction de la visibilité permet d'obtenir un accès flexible entre les différents ensembles de paquetages (Paquetage 1, Paquetage n) stockés dans le même espace nom, mais présente quelques inconvénients. La plate-forme classique ou spécifique supporte mal la notion de sous-paquetages. Les classes client d'une application de grande taille doivent être réparties entre différents sous-paquetages qui représentent uniquement pour la machine virtuelle des paquetages différents. Pour partager les ressources entre ces sous-paquetages, ces ressources sont nécessairement déclarées publiques, les rendant ainsi visibles de tout autre paquetage. Il est ainsi difficile d'organiser de façon claire le paquetage d'applications différentes pour des applications de grande taille et d'obtenir des pare-feu entre les applications.

Dans le cas de la plate-forme classique, sur une station, le respect de la confidentialité des ressources, telle que déclarée dans les programmes sources, repose sur des vérifications dynamiques. Pour effectuer ces vérifications dynamiques, la machine virtuelle doit posséder toutes les informations concernant les déclarations de restriction d'accès pour chaque classe, méthode ou attribut, ce qui ne peut être possible dans le cas de l'espace mémoire disponible sur la machine virtuelle embarquée (onplatform) utilisant le pseudocode intermédiaire ou byte code prélié de la machine virtuelle hors plate-forme (offplatform). Dans le cas d'une plate-forme spécifique d'un objet portable, les restrictions d'accès peuvent se vérifier statiquement lors de la compilation et peuvent être vérifiées à nouveau par le vérifieur de la partie hors plate-forme. Il est en effet supposé que ce qui est chargé sur la plate-forme spécifique de l'objet portable est correct, car aucune vérification ne peut être effectuée sur la plate-forme spécifique du fait de la perte des informations lors de la phase de conversion. De plus il n'est pas garanti qu'un paquetage ne puisse pas être étendu ou modifié par de nouvelles classes venant de sources étrangères, ce qui peut détruire toute la sécurité basée sur l'utilisation de paquetages privés.

Le deuxième moyen procuré par la machine virtuelle d'une plate-forme classique est la notion de séparation des espaces nom. Avec le schéma de liaison dynamique d'une plate-forme classique, les classes peuvent être chargées, à partir de répertoires du système de fichier local, appelé "ClassPath" préalablement déclaré comme constituant l'emplacement des classes systèmes formant la plate-forme API standard, ou à partir d'autres répertoires du système de fichier local ou de serveurs éloignés. Les classes client d'une application, extérieures au "ClassPath", doivent être chargées par un chargeur de classe spécifiquement programmé. Cette caractéristique est particulièrement utilisée pour le chargement des applications "Java" par des navigateurs habilités "Java". Ainsi, les applications venant de différentes sources, sont chargées par des chargeurs de classes différents. Pour chaque localisateur, communément appelé URL

(Uniform Resource Locator), une instance spécifique de classe "chargeur de classe d'application" est utilisée. Le nom externe d'une classe est l'assemblage <Nom Du Paquetage > + <Nom De La Classe>. Quand une classe est chargée, elle est stockée dans une table de classe interne, et une référence relative au chargeur de classe utilisé pour charger cette classe est ajoutée en préfixe du nom de paquetage de la classe. Le nom de la classe est alors <<Référence du Chargeur de Classe> + <Nom Du Paquetage> + <Nom De La Classe>>. Ainsi des classes déclarées comme appartenant au même paquetage mais chargées par des chargeurs de classes différents ne sont pas perçues par la machine virtuelle comme appartenant au même paquetage.

Lors de la résolution d'une référence, la politique habituelle des chargeurs de classe est de rechercher d'abord dans les classes système et en cas d'échec, de rechercher parmi les fichiers de classe présents à l'emplacement où le chargeur peut charger les classes. Selon ce principe de fonctionnement du chargeur, une application ne peut directement accéder aux classes clients d'une autre application chargée par un chargeur de classes différent. Une application peut accéder à toutes les ressources publiques des classes publiques du ClassPath, mais les classes du ClassPath ne peuvent accéder directement aux classes d'une application bien qu'elles puissent faire référence à des instances de classes clients de l'application en les convertissant en un type public défini dans le "Classpath". Une application ne peut étendre ou modifier des paquetages de classes système du Classpath ou de toutes autres applications chargées par des chargeurs de classes différents. L'utilisation de la séparation des espaces nom en insérant une référence du chargeur de classe dans le nom de la classe, ajoutée au typage complet fourni par le langage "Java", permet de procurer un pare-feu efficace entre les applications. Le mécanisme inné de séparation de nom d'espace permet facilement le chargement de nouvelles applications. Les applications peuvent échanger des objets par l'entremise des classes spécifiquement programmées à cette fin et localisées dans le

"Classpath". Une application peut utiliser un objet venant d'une autre application chargée par un chargeur de classe différent, si cet objet peut être changé en un type public défini dans le "Classpath".

Malheureusement ce mécanisme de séparation de nom, basé sur une machine virtuelle classique et son processus de liaison dynamique ne peut être effectué sur une plate-forme spécifique. La liaison dynamique ne peut être effectuée par la machine virtuelle d'une plate forme spécifique, celle-ci nécessitant un espace mémoire permettant le stockage de fichier de classe d'une plate-forme "Java" classique, présentant des références non résolues. Dans une plate-forme spécifique, les classes systèmes des API et les classes clients des applications ne sont pas isolées dans des espaces de nommage particuliers.

L'objet de la présente invention est de proposer une solution procurant les avantages de la séparation de nom dans le cadre de systèmes embarqués possédant une machine virtuelle en deux parties, le schéma de liaison statique étant effectué par la partie hors plate-forme.

Dans le contexte de systèmes embarqués, tels que par exemple les terminaux de paiement, multi-application, il est nécessaire d'avoir des pare-feu performants entre les applications "Java" fournies par différentes sources, telles que différents systèmes de paiement (Visa, Mastercard...). Il peut être utile de permettre une coopération flexible entre les applications venant des différentes sources. Le système embarqué doit aussi être facilement mis à jour en téléchargeant de nouvelles applications ou de nouveaux modules utilitaires ou encore des mises à jour sans perturber les applications déjà chargées.

Un premier but de l'invention est de proposer un procédé de chargement permettant d'obtenir des pare-feu robustes entre les applications tout en autorisant une coopération entre les applications et la possibilité de faire évoluer les applications ou de charger d'autres applications.

Ce but est atteint par le fait que le procédé de chargement d'applications sur un système embarqué selon l'invention, comprenant un

environnement d'exécution incluant une machine virtuelle comprenant un interpréteur de langage de type pseudocode intermédiaire, des interfaces de programmation d'application (API), à partir d'une station sur laquelle le code source de l'application est écrit, compilé en pseudocode par un compilateur , vérifié par un vérificateur, converti par un convertisseur et chargé par un chargeur, se caractérise en ce que

- la conversion comprend la réalisation de la liaison statique d'une pluralité d'ensembles de paquetages destinés à être stockés dans le même espace nom sur le système embarqué, appelés modules, en attribuant un identificateur à chaque module (MID), et un numéro de référence à chaque classe, à chaque méthode et à chaque attribut encapsulés dans les classes du module,

- la référence à une méthode ou un attribut, dans le pseudocode lié d'un module, étant codée sur trois multiplets constitués par un indicateur indiquant la référence à une classe interne ou externe au module, le numéro de la classe et soit le numéro de la méthode soit le numéro de l'attribut,

- les modules chargés sont un ou plusieurs modules d'interface de programmation d'application, appelé Module API, comprenant des classes système ou des modules de services correspondant chacun à une application, une référence à une classe externe étant systématiquement interprétée par la machine virtuelle comme une référence à un module d'interface de programmation d'application.

Selon une autre particularité, le chargement des modules sur le système embarqué comprend la mémorisation d'une part d'au moins un tableau de représentation des modules, le numéro associé, par le lieur compris entre 0 et n, à un module constituant l'index dudit module dans le tableau, et d'autre part d'une table mémorisant l'association de l'index du tableau de représentation à l'identificateur (MID) dudit module, ledit tableau et la table étant en mémoire programmable non volatile, une référence externe à un module externe dans le pseudocode étant interprétée par

l'interpréteur de la machine virtuelle comme constituant un index d'accès au module équivalent à celui du tableau des modules.

Selon une autre particularité, le chargement comprend la mémorisation, pour chaque module, d'un tableau de représentation de ses classes, comprenant une référence à l'index de son module et, pour chaque classe, un tableau de représentation des attributs et des méthodes.

Selon une autre particularité, les modules sont référencés dans un tableau de modules unique, les classes système sont contenues dans un module API unique, toute référence à une classe externe dans le pseudocode différente de n sera interprétée par la machine virtuelle comme une référence audit module API.

Selon une autre particularité, les classes étant déclarées publiques, ou en paquetage privé, les attributs et méthodes étant déclarés protégés, en paquetage privée ou en classe privée, la numérotation des classes s'effectue suivant l'ordre classes publiques puis classes en paquetages privés, la numérotation des attributs ou méthodes est effectuée par le convertisseur suivant l'ordre attribut ou méthode public, protégé, en paquetage privé et en classe privée.

Selon une autre particularité, les classes système sont contenues dans plusieurs modules API chargeables séparément, le chargeur maintient dans la mémoire non volatile programmable deux tableaux de représentation des modules et deux tables d'association MID/IMi correspondantes, l'un pour les modules API et l'autre pour les modules non-API, le chargeur chargeant les modules dans l'un des deux tableaux selon la nature du module spécifié dans l'entête de celui-ci, toute référence externe d'un module du tableau de module étant interprétée comme une référence à l'index du module API.

Selon une autre particularité, la liaison statique d'un module est effectuée de telle sorte que la référence à une classe externe à un module non API dans le pseudocode intermédiaire est un index dans un tableau de l'entête du module, dont chaque entrée est un identificateur (MID) d'un module API référencé, le chargement dudit module sur la plate-forme cible

comprenant le remplacement de ladite référence par le numéro de l'index du module API obtenu à partir de l'identificateur (MID) de la table d'association des modules API.

Un autre but de l'invention est de proposer un système embarqué correspondant.

Ce but est atteint par le fait que le système embarqué selon l'invention, comprenant une machine virtuelle et une plate-forme API incluant des interfaces de programmation d'application, une mémoire non volatile fixe, une mémoire non volatile programmable ou modifiable, et une mémoire vive, se caractérise en ce que la mémoire non volatile programmable comprend au moins un module API comprenant des classes système et des modules de services, au moins un tableau de représentation des modules, dans lequel les modules sont indexés et une table associant l'index d'un module du tableau de représentation à l'identificateur dudit module, chaque module comprenant un tableau de représentation des classes, dans lequel les classes sont indexées et dans lequel chaque classe présente une référence à l'index de son module, chaque classe comprenant un tableau de représentation des attributs et des méthodes, dans lesquels les attributs et méthodes sont indexés, la référence à une méthode ou un attribut étant codée sur au moins trois multiplets correspondant à une référence à une classe interne ou externe au module, une référence externe au module constituant l'index du module API dans le tableau de module, un numéro de classe correspondant à l'index de la classe dans la table de représentation des classes du module, et un numéro de méthode ou d'attribut correspondant à l'index de la méthode ou de l'attribut dans le tableau de représentation des méthodes ou attributs de la classe du module.

Selon une autre particularité, le système embarqué comporte des moyens de comparaison du premier multiplet des trois multiplets de codage de référence à une méthode ou à un attribut avec une valeur déterminée n pour décider s'il s'agit d'une classe interne ou externe.

Selon une autre particularité, le système embarqué comprend un module principal comprenant le programme principal du système.

Selon une autre particularité, les classes sont indexées suivant l'ordre classes publiques puis classes en paquetages privés, et les attributs ou méthodes suivant l'ordre attribut ou méthode public, protégé, en paquetage privé et en classe privée.

Selon une autre particularité, la mémoire non volatile programmable comprend plusieurs modules API comprenant des classes système, deux tableaux de représentation des modules, l'un pour les modules API et l'autre pour les modules non-API et le module principal, et deux tables d'association MID/IMi correspondant chacune à un tableau de représentation des modules.,

Selon une autre particularité, le système embarqué comprend une classe gestionnaire d'accès "Access manager" d'un module API comprenant une méthode permettant de créer une instance d'un module de service, par l'intermédiaire du module principal, ladite classe présentant une protection lui interdisant d'avoir plus d'une instance.

Un autre but de l'invention est de proposer un procédé d'exécution d'une application présente sur un système embarqué multi-application.

Ce but est atteint par le fait que le procédé d'exécution d'une application d'un système embarqué multi-application, comprenant un environnement d'exécution incluant une machine virtuelle comprenant un interpréteur de langage de type pseudocode intermédiaire, et des interfaces de programmation d'applications (API), se caractérise en ce que, lors de l'exécution du pseudocode intermédiaire d'un module de service, correspondant à une application, référencée dans un tableau de module, la référence à une méthode ou un attribut dans le pseudocode, codée sur au moins trois multiplets correspondant à une référence à une classe interne ou externe au module, un numéro de classe et un numéro de méthode ou d'attribut, une référence externe au module est interprétée par la machine

virtuelle comme une référence à l'index d'un module API du tableau du ou des modules API.

Selon une autre particularité, sur réception d'une demande d'exécution d'un module de service présentant un identificateur, l'environnement d'exécution accède à la classe d'entrée d'un module principal comprenant le programme principal du système, le module principal installe une instance d'une classe spéciale "Access Manager", d'un module API, gérant l'accès à un module de service et utilise une méthode de cette classe permettant de créer une instance de la classe d'entrée du module de service demandée, par l'intermédiaire d'une table d'association de l'identificateur à l'index du module dans un tableau dans lequel le module est référencé, l'instance étant retournée par la méthode au programme principal.

D'autres particularités et avantages de la présente invention apparaîtront plus clairement à la lecture de la description ci-après faite en référence aux dessins annexés dans lesquels :

- la figure 1 représente de façon schématique les différents éléments nécessaires pour le chargement d'un objet portable selon un premier mode de réalisation ;

- la figure 2 représente de façon schématique les différents éléments nécessaires pour le chargement d'un objet portable selon un deuxième mode de réalisation ;

- la figure 3 représente la représentation interne d'un module ;

- la figure 4a représente le schéma classique d'une carte à puce ;

- la figure 4b représente le système nécessaire à la constitution d'une machine virtuelle embarquée sur une carte à puce selon l'art antérieur;

- la figure 4c représente la structure des classes d'une application.

Le procédé sera décrit, en liaison avec les figures 1 à 3, de manière non limitative, dans le cas de la mise en œuvre de l'invention dans un système embarqué, par exemple de type spécifique constitué par une carte à puce ou un objet portable similaire. La désignation byte code ou

programme de type byte code recouvre tout pseudocode ou programme intermédiaire.

L'objet portable constitue par exemple une carte à puce et présente une structure similaire de celle décrite précédemment en référence aux figures 4a et 4b, et comprend notamment une mémoire RAM, ROM et EEPROM. La plate-forme spécifique (60) et une station classique (80) sont représentées sur la figure 1. La plate-forme spécifique (60) possède en ROM, un environnement d'exécution (RE) comprenant des API (62) et une machine virtuelle (61) embarquée ("onplatform"). La plate-forme spécifique (60) est représentée sur la figure 1, comme comprenant l'ensemble des mémoires ROM et EEPROM. Il est à noter que la plate-forme spécifique (60) désigne plus particulièrement l'environnement d'exécution (RE) et les éléments présents en mémoire EEPROM. L'objet portable possède en ROM un système d'exploitation (Operating System) (63). Les API (62), présentes en mémoire ROM, constituent les API de base de la plate-forme API, chargées avec la machine virtuelle embarquée pour le fonctionnement de celle-ci.

La partie (90) hors objet portable de la machine virtuelle comprend un vérifieur (91) de pseudocode intermédiaire, un convertisseur (92) et éventuellement un signeur (94). Le signeur délivre une signature pour valider le passage par le vérifieur et le convertisseur. Cette signature sera vérifiée par l'objet portable au moment du chargement. Le chargement en EEPROM d'applications ou de nouvelles API pour compléter les API de base s'effectue par un chargeur qui peut être composé de deux parties, une partie hors objet portable pouvant être installée dans la machine virtuelle hors objet portable (90), appelée téléchargeur (93), et une partie sur la plate-forme spécifique, appelée chargeur (68).

Selon un premier mode de réalisation, la plate-forme spécifique (60) comprend deux modules spéciaux, un module API (65) et un module principal (66). Les autres modules sont appelés modules de services (67). Chaque module correspond à un ensemble de paquetages qui sera stocké

dans le même espace nom. La plate forme API désigne les API de base (62) et l'ensemble des classes système qui définissent le module API (65) ou module de la plate-forme API. Le module principal comprend la classe principale définissant le programme principal. Chaque module, excepté le module API (65), possède une classe unique, particulière, appelée "Entry Class", qui constitue le point d'accès au module. Pour le module principal, cette "Entry Class" est la classe principale (CP), celle qui contient une méthode statique appelée "main". Pour les modules de services, c'est une classe avec seulement un constructeur sans paramètres et implémentant une interface publique spéciale, appelée "service" définie dans la plate-forme API. Le chargement d'une application correspond au chargement d'un module de service. Chaque module reçoit un identificateur spécifique. Un tel identificateur, qui est appelé MID, peut par exemple être un nombre, une chaîne de caractères, ou un tableau. A titre d'exemple, l'identificateur est une chaîne de caractères.

Quand ils sont chargés dans la plate-forme, par des mécanismes de téléchargement distincts de la machine virtuelle de la plate-forme spécifique, les modules reçoivent un nombre, compris entre 0 et n. Ainsi, selon cette convention, n+1 modules au plus peuvent être présents sur la plate-forme spécifique. Le téléchargeur (93) de module avec le chargeur (68) maintient, lors du chargement de nouveaux modules de service, un tableau (TRM) (69) de représentation des modules. Le numéro associé à un module est l'index (IM) de ce module dans le tableau. Le chargeur (68) maintient également une table (70) associant l'index (IM) à l'identificateur (MID) de chaque module. Le module API reçoit systématiquement pour index IM_0 le numéro 0, et le module principal pour index IM_1 le numéro 1. L'entête (header) de chaque module comprend un indicateur permettant au chargeur de déterminer la nature du module, "principal", modules "de service" ou module "API".

Le chargeur (68) ne peut charger que les modules autorisés à résider sur l'objet portable, c'est à dire uniquement les modules présentant

une signature connue de l'objet portable. Le chargeur (68) comporte donc des moyens de vérifier la signature d'un module reçu, de la comparer avec la signature connue de l'objet portable et en cas de comparaison négative de bloquer le chargement.

De manière classique, tel que défini dans l'art antérieur cité précédemment, le programme source (81) d'une application est écrit puis compilé par un compilateur (82) et ensuite vérifié par le vérifieur (91).

La liaison statique, réalisée dans le convertisseur (92) par un composant dit lieur (linker) du convertisseur, va résoudre des références symboliques en attribuant

- un numéro (NCl) à chaque classe d'un module,
- un numéro (NM) pour chaque méthode dans une classe et
- un numéro (NA) pour chaque attribut dans une classe.

Chacun de ces numéros (NCl, NM, NA) est compris entre 0 et n et peut ainsi être représenté sur un byte ou multiplet. A titre d'exemple, chacun de ces nombres sera compris entre 0 et 255 (n=255). La référence à une méthode ou un attribut d'une classe sera ainsi codée dans le pseudocode lié des méthodes du module sur deux multiplets (bytes) ou octets. Le pseudocode contiendra ces deux multiplets < NCl> pour la classe et < NA> pour un attribut ou <NM> pour une méthode.

Suivant la figure 3, la représentation interne d'un module API (65), d'un module principal (66) ou d'un module de service (67), contiendra un tableau (TRC) de représentation de classes ; le numéro (NCl) associé par le lieur, hors du système embarqué, à chaque classe est l'index (ICi) de la représentation de cette classe dans le tableau (TRC). Chaque classe présente également une référence à l'index (IM_i) de son module. De la même manière, la représentation de chaque classe contient un tableau des représentations de méthodes (TRMe) et un tableau de représentation des attributs (TRA) appartenant à la classe. Le numéro (NM), associé par le lieur, hors du système embarqué, à chaque méthode est l'index (IMi), de la représentation de cette méthode dans le tableau (TRMe), et le numéro (NA),

associé par le lieur, hors du système embarqué, à chaque attribut est l'index (IAi), de la représentation de cet attribut dans le tableau (TRA).

A titre d'exemple, nous voulons qu'un module puisse référer uniquement à ses propres classes et aux classes systèmes de la plate-forme API, les classes système correspondant aux classes du "ClassPath" d'une plate-forme classique. Selon l'invention, pour permettre la distinction entre une référence à une classe interne au module et la référence à une classe système (ou externe au module), un indicateur interne (II) ou externe (IE) est ajouté à la référence à une méthode ou à un attribut. La référence résolue est alors codée sur trois multiplets : <IE/II> <NCl> < NM> ou <IE/II> <NCl> <NA>

Selon une convention posée, pour la valeur n du premier multiplet <IE/II> la valeur 255 d'après notre exemple, correspond à une référence interne <II> au module et toute autre valeur pour le premier multiplet correspond à une référence externe <IE> au module.

Le lieur du convertisseur (92) de la machine virtuelle hors objet portable (90), relie en premier le module API (65), qui ne possède pas de références externes <IE> dans son pseudocode, et produit une implantation ou agencement, correspondant à un plan de noms symboliques de ses classes et de leurs méthodes et attributs. Lors de la mise en liaison des autres modules, cette implantation sera utilisée pour établir les références externes à des classes systèmes du module API (65).

Suivant notre convention des multiplets (bytes) encodant les références, il peut y avoir au plus 256 (n+1) classes dans le module API et 256 classes dans chaque module supplémentaire.

Lors de l'exécution d'un module de service, quand la machine virtuelle (61) trouve une référence à une méthode <NM> ou un attribut <NA> dans le pseudocode, en connaissant la classe <NCl> où se trouve cette référence, elle peut retrouver directement l'index <IMi> du module concerné, celui-ci correspondant à la référence externe (IE) ou interne (II). Toute référence externe <IE> dans le pseudo code d'un module de service sera

systématiquement interprétée par la machine virtuelle comme une référence au module API. Un module de service ou le module principal ne peut pas référer aux classes de tout autre module excepté celles du module API. Les classes systèmes de ce module API ne peuvent pas référer aux classes d'un module de service ou du module principal. La référence interne à une classe d'un module, correspondant à la valeur n pour le premier multiplet, ne nécessite aucune connaissance a priori de l'espace nom qui sera attribué au module. Le fait de ne pas définir a priori d'espace de nommage fixe lors de la phase de conversion permet d'accélérer la résolution des références et de déterminer l'espace de nommage d'un module lors du chargement, postérieurement à la phase de conversion. La machine virtuelle, lors de l'interprétation d'une référence à un attribut ou à une méthode dans le pseudocode, utilise les trois index <IE/II> <NCI> < NM> ou <IE/II> <NCI> <NA> en cascade. L'espace mémoire du module étant déterminé, l'index <NCI> détermine l'entrée désirée dans le tableau des classes (TRM) du module, puis le dernier index < NM> ou <NA> donne l'entrée désirée dans le tableau des méthodes (TRMe) ou le tableau des attributs (TRA).

Le module API comprend une classe spéciale (64), appelée classe "gestionnaire d'accès" ou "Access Manager", qui comprend une méthode native (getServiceInstance) dont le rôle est de rendre un objet instance de la classe d'entrée du module de service demandé, à partir de l'identificateur (MID) du module. Cette méthode utilise la table (70) d'association MID/Imi pour connaître l'index du module demandé dans le tableau de module (69) puis crée une instance de la classe d'entrée de ce module, instance qui est retournée par la méthode. Selon l'invention la classe "Access Manager" (64) est protégée par construction par une méthode consistant à interdire que cette classe ait plus d'une instance. Cette méthode (getServiceInstance) appartient au programme principal contenu dans le module principal. Le module principal qui sera activé en premier à utilisation de l'objet portable crée une instance et une seule de la classe "Access Manager", ce qui lui

permet d'utiliser la méthode getServiceInstance, mais interdit à tout autre service de créer une autre instance pour utiliser cette méthode.

En fonctionnement, de la même façon que sur une plate-forme classique, l'environnement d'exécution (RE) accède à la classe d'entrée (EC) du module principal et active sa méthode d'entrée (main). Le module principal, étant le premier activé, procède à l'installation d'une instance de la classe "Access manager" avant que tout autre service le fasse, puisque pour activer d'autres services, le module principal doit déjà posséder une telle instance de la classe accès.

Ce simple dispositif permet de reproduire l'effet de protection lié au concept d'espace de nommage d'une plate-forme classique. Le simple fait de charger un module de service dans le tableau des modules et que la présence dans le pseudocode de toutes référence externe soit interprétée par la machine virtuelle comme une référence au module API, rend ce module complètement inaccessible directement par les autres modules, créant ainsi un pare-feu total.

Ce premier mode de réalisation permet d'apporter les avantages de pare-feu procuré par la séparation des espaces nom dans le contexte d'une machine virtuelle en deux parties. Toutefois ce mode de réalisation se révèle peu flexible et présente deux inconvénients.

Premièrement, il empêche toute modification ou extension des classes systèmes avec des modules déjà préliés. Une architecture "Java" classique permet de modifier et d'étendre les classes de la plate-forme API sans avoir d'impact sur les classes déjà compilées de modules supplémentaires. Mais dans le mode de réalisation décrit précédemment, toute modification des classes système, même invisible pour des modules étrangers, modifierait l'agencement de la plate-forme API et nécessiterait de modifier le pseudocode prélié de chaque module déjà lié avec une version antérieure de l'agencement et en conséquence l'interpréteur.

Deuxièmement, les modules préliés sont supposés être portables entre les différentes plates-formes ou terminaux embarqués, ce qui impose

que chacune de ces plates-formes doit avoir le même agencement que la plate-forme API, ce qui interdit l'utilisation de toute extension propriétaire.

Afin de remédier partiellement à ces inconvénients, une variante du premier mode de réalisation, consiste à imposer que, dans la numérotation de l'implantation, les classes publiques viennent en premier avant les classes en paquetage privé. De plus, les méthodes publiques ou les attributs publics viennent avant ceux qui sont protégés et ceux qui sont en paquetages privés et en classes privées. Ceci permet d'ajouter librement de nouvelles classes publiques dans le module API (65).

La figure 2 représente un deuxième mode de réalisation permettant l'évolution de la plate-forme API. La plate-forme API est constituée de plusieurs modules API (100) pouvant être chargés séparément, au lieu d'être constituée d'un module API unique.

Dans ce mode de réalisation, le téléchargeur (93) et la machine virtuelle partagent deux tableaux de modules et deux tables d'association MID/index au lieu d'un de chaque, un tableau (101) et une table d'association (102) pour les modules API et un tableau (103) et une table d'association (104) pour les modules non-API, correspondant au module de service (67) et au module principal (66).

Chaque module présente dans son entête un indicateur indiquant sa nature "Service" ou "API" permettant au chargeur de charger le module dans le tableau (101) de modules API ou dans le tableau (103) de modules non-API. Cet indicateur est placé dans l'entête du module lors de la phase de compilation par le convertisseur.

Le pare-feu constitué par la séparation de l'espace nom est présent uniquement entre les modules non-API. Toute référence externe à un module de service sera interprétée par l'interpréteur de la machine virtuelle embarquée comme un index du tableau du module API.

Les modules non-API seront numérotés de 0 jusqu'à 255 au plus, dans l'exemple où n=255. 0 est par exemple l'index du module principal (66). Les modules API (100) seront numérotés de 0 à 254 au plus, 0 étant par

exemple l'index d'un module dit API primaire, qui contient toutes les méthodes natives. Conformément à la convention décrite précédemment, ceci permet au plus, 255 (n) modules différents dans la plate-forme API. La référence à une méthode ou attribut dans le pseudocode est :

<< IE/II> <NCI > < NM/NA>>

La valeur 255 (n) pour le premier multiplet (byte) indiquera, comme dans le premier mode de réalisation, une référence interne au module. Chaque valeur différente de 255 indiquera une référence externe à un module spécifique (100) du tableau de module API de la plate-forme API.

Après la réalisation de la liaison par le lieur (92) hors plate-forme, le pseudocode d'un module comporte une entête présentant un tableau de modules référencés utilisé pour lier le module courant. Ce tableau de modules référencés comprend au plus 255 entrées, chaque entrée correspondant à l'identificateur (MID) d'un module API (100). Le premier multiplet (byte) d'une référence externe dans le pseudocode sera alors un index dans ce tableau. Lors du chargement sur la plate-forme d'un module non API (67, 66), les numéros d'index associés à des modules API (100) seront connus et ainsi chaque premier octet d'une référence externe sera remplacé par le numéro d'index associé au module API référé en utilisant la table (102) d'association MID/IMi des modules API (100). Ce remplacement est la seule opération de liaison réalisée sur la plate-forme spécifique, par le chargeur (68), la table (102) d'association MID/IMi étant utilisée uniquement pour réaliser cette opération de liaison.

A titre d'exemple, supposons que dans le pseudocode d'un module de service "TEST", nous avons la référence à un numéro de méthode 5 du numéro de classe 7 d'un module API dont l'identificateur (MID) est "F00". Supposons de plus que "F00" est l'identificateur du quatrième module API externe trouvé référencé dans le module de service "TEST". La référence dans le pseudocode est alors constituée par les trois valeurs suivantes : 3, 7, 5.

Le numéro 3 correspond au quatrième index dans le tableau de modules référencés présent dans l'entête du module, venant en tête du pseudocode, la valeur de cette entrée étant l'identificateur (MID) "FOO". Supposons que lors du chargement du module API "FOO, l'index interne 34 lui ait été attribué sur la plate-forme cible dans la table d'association (102) des modules API. Alors, le chargeur (68), à l'aide de la table d'association (102) modifie la référence dans le pseudocode du module de service "TEST" pour devenir : 34, 7, 5.

Lors de l'exécution du pseudocode d'un module, une référence externe au module est systématiquement interprétée par la machine virtuelle comme une entrée dans la table de module API. Les modules du tableau de module non API restent invisibles les uns des autres ainsi que par rapport aux modules API. Ce simple dispositif permet de reproduire l'effet de protection lié au concept d'espace de nommage d'une plate-forme classique. Le simple fait de charger un module dans le tableau de module non API le rend complètement inaccessible directement par les autres modules, créant ainsi un pare-feu total.

Le tableau (101) de modules API comprend un module spécifique (105), appelé module "API Access" qui comprend une méthode native (getServiceInstance) dans une classe "gestionnaire d'accès" ou "Access Manager" dont le rôle est de rendre un objet instance de la classe d'entrée du module de service demandé. Cette méthode utilise la table (104) d'association MID/IMi pour connaître l'index du module de service demandé dans le tableau (103) de modules non-API puis crée une instance de la classe d'entrée de ce module qui est retournée par la méthode au programme principal. La politique de sécurité préconisée est de faire de la classe "Access Manager" une classe protégée dont le constructeur et les méthodes sont déclarés protégés. De plus, le module "API Access" (105) comprend une protection consistant à interdire que la classe "Access Manager" ait plus d'une instance. Cette méthode est réservée au programme principal contenu dans le module principal (66). Le module principal qui est

activé en premier crée une instance du module Access manager, ce qui lui permet d'utiliser la méthode getServiceInstance, mais interdit à tout autre service de créer une autre instance pour utiliser cette méthode. Ainsi le module principal pourra créer des instances de services.

Plusieurs méthodes peuvent être utilisées pour obtenir cette protection consistant à interdire que la classe "Access manager" n'ait qu'une seule instance. Le constructeur de la classe peut par exemple bloquer la demande de création d'instance lorsqu'il en existe déjà une et lance une exception de sécurité. En fonctionnement, l'environnement d'exécution (RE) accède à la classe d'entrée du module principal (66) et active sa méthode d'entrée (main). Le module principal étant le premier activé, procède à l'installation d'une instance de la classe "Access Manager" du module Access avant que tout autre service le fasse.

Afin de permettre à un module de service d'activer un autre module de service, cette politique stricte de sécurité peut être modifiée en ajoutant à la classe "Access Manager" du module API Access (105) des classes publiques permettant à tout module d'y faire des requêtes. Ces requêtes seront traitées et contrôlées par l'unique instance créée par le module principal. Ces classes publiques comprennent notamment une méthode statique permettant d'obtenir l'unique instance. Un module ayant accès à l'objet instance de la classe "Access Manager" pourra activer un autre module de service et l'utiliser, mais il ne pourra pas référencer directement ses classes, ses méthodes ou ses attributs sans être repéré par la machine virtuelle, étant donné que toute référence externe dans le pseudocode est une référence interne au module ou une référence externe à un module API.

Pour une réalisation simple de cette solution, il est nécessaire de ne pas avoir de références circulaires parmi les modules API. En conséquence, la fermeture transitive de la relation "réfère à"("refers to") doit être un ordre strict partiel sur un jeu de modules. Il est ainsi possible de concevoir dans le lieur du convertisseur (92) une stratégie simple pour lier et produire l'agencement des modules API en traitant en premier les éléments

minimums non encore liés. Il est possible de suivre la même stratégie basée sur un ordre partiel pour le téléchargement des modules API, de telle sorte que lors du téléchargement d'un module M, tous les modules auquel il se réfère aient déjà été téléchargés et qu'un numéro leur aient été assigné. L'assignation de l'index interne sur la plate-forme cible se fait par le chargeur (68) de module en assignant l'index n-1 à l'API d'ordre n. Un module API ne peut référer à un autre API module d'index supérieur.

L'utilisation de ce système de double tableau de modules (101, 103) et de table d'association (102, 104), permet de remplacer facilement un module API unique par plusieurs modules API chargeables séparément. Le remplacement d'un module API unique par plusieurs modules API permet d'étendre la plate-forme API avec de nouveau modules, sans modifier l'assemblage des modules déjà chargés, sans changer la sécurité offerte par les pare-feu. Bien entendu, ces deux modes de réalisation ne sont pas compatibles ; les modules doivent être préliés spécifiquement pour l'un ou l'autre des modes de réalisation, le pseudocode relatif à un des modes de réalisation n'étant pas portable sur une plate-forme implémentant l'autre mode de réalisation. De plus, l'interpréteur de la machine virtuelle diffère d'un mode de réalisation à l'autre. Dans le premier mode de réalisation, la machine virtuelle ne manipule qu'un seul tableau et une table d'association : le premier multiplet d'une référence sera interprété par la machine virtuelle comme une référence interne pour toute valeur égale à n et comme une référence externe à l'unique module API pour toute valeur différente de n. Dans le second mode de réalisation, la machine virtuelle manipule deux tableaux et deux tables d'association : le premier multiplet d'une référence dans le pseudocode sera interprété par la machine virtuelle comme une référence interne au module pour toute valeur égale à n et toute valeur différente de n sera prise directement comme index dans le tableau de module API. Dans les deux modes de réalisation l'interpréteur de la machine virtuelle comprend des moyens de comparaison du premier multiplet des trois multiplets de codage d'une référence à une méthode ou à un attribut

avec une valeur déterminée n pour décider s'il s'agit d'une classe interne ou externe au module. La numérotation des modules API peut être déterminée au moment du chargement pour fixer définitivement et de manière très simple les références externes dans le pseudocode.

Les mêmes mécanismes sont utilisés pour manier les deux types de modules, bien que la façon dont ils sont utilisés et la sécurité procurée soient tout à fait différentes. Tout module peut accéder librement aux modules API puisque leurs classes sont des classes système. L'utilisation de l'approche modulaire est utilisée avec les modules services pour procurer un pare-feu rigoureux pour protéger ces modules de tout accès direct.

Le procédé selon l'invention peut être réalisé sur tous types d'objet portable présentant de faibles ressources, tel que par exemple, 16 Ko de mémoire ROM, 8ko de mémoire EEPROM et 256 k de mémoire RAM.

D'autres modifications à la portée de l'homme de métier font également partie de l'esprit de l'invention.

REVENDICATIONS

1. Procédé de chargement d'applications sur un système embarqué comprenant un environnement d'exécution incluant une machine virtuelle comprenant un interpréteur de langage de type pseudocode intermédiaire, des interfaces de programmation d'application (API), à partir d'une station sur laquelle le code source de l'application est écrit, compilé en pseudocode par un compilateur (82), vérifié par un vérificateur (91), converti par un convertisseur (92), et chargé par un chargeur (93, 68), caractérisé en ce que

- la conversion comprend la réalisation de la liaison statique d'une pluralité d'ensembles de paquetages destinés à être stockés dans le même espace nom sur le système embarqué, appelés modules, en attribuant un identificateur à chaque module (MID), et un numéro de référence à chaque classe (NCI), à chaque méthode (NM) et à chaque attribut (NA) encapsulés dans les classes du module,

- la référence à une méthode ou un attribut, dans le pseudocode lié d'un module, étant codée sur trois multiplets constitués par un indicateur indiquant la référence à une classe interne (II) ou externe (IE) au module, le numéro de la classe (NCI) et soit le numéro de la méthode (NM) soit le numéro de l'attribut (NA),

- les modules sont un ou plusieurs modules d'interface de programmation d'application comprenant des classes système ou des modules de services correspondant chacun à une application, une référence (IE) à une classe externe étant systématiquement interprétée par la machine virtuelle comme une référence à un module d'interface de programmation d'application.

2. Procédé de chargement d'applications sur un système embarqué selon la revendication 1, caractérisé en ce que le chargement des modules sur le système embarqué comprend la mémorisation d'une part d'au moins un tableau (69, 101, 103) de représentation des modules, le numéro (IMi) associé, par le lieur compris entre 0 et n, à un module constituant l'index

dudit module dans le tableau, et d'autre part d'une table (70, 102, 104) mémorisant l'association de l'index du tableau de représentation à l'identificateur (MID) dudit module, ledit tableau et la table étant en mémoire programmable non volatile, une référence externe (IE) à un module externe dans le pseudocode étant interprétée par l'interpréteur de la machine virtuelle comme constituant un index d'accès au module équivalent à celui du tableau des modules.

3. Procédé de chargement d'applications sur un système embarqué selon la revendication 2, caractérisé en ce que le chargement comprend la mémorisation, pour chaque module, d'un tableau de représentation (TRC) de ses classes, comprenant une référence à l'index de son module et, pour chaque classe, un tableau de représentation (TRA) des attributs et des méthodes (TRMe).

4. Procédé de chargement d'applications dans un système embarqué selon la revendication 2, caractérisé en ce que les modules sont référencés dans un tableau de modules unique, les classes système sont contenues dans un module API unique, toute référence à une classe externe dans le pseudocode différente de n sera interprétée par la machine virtuelle comme une référence audit module API.

5. Procédé de chargement d'applications dans un système embarqué selon la revendication 4, caractérisé en ce que les classes étant déclarées publiques, ou en paquetage privé, les attributs et méthodes étant déclarés protégés, en paquetage privée ou en classe privée, la numérotation des classes s'effectue suivant l'ordre classes publiques puis classes en paquetages privés, la numérotation des attributs ou méthodes est effectuée par le convertisseur (92) suivant l'ordre attribut ou méthode public, protégé, en paquetage privé et en classe privée.

6. Procédé de chargement d'applications sur un système embarqué selon la revendication 2, caractérisé en ce que les classes système sont contenues dans plusieurs modules API chargeables séparément, le chargeur (68) maintient dans la mémoire non volatile programmable deux

tableaux de représentation des modules et deux tables d'association MID/Imi correspondantes, l'un pour les modules API et l'autre pour les modules non-API, le chargeur chargeant les modules dans l'un des deux tableaux selon la nature du module spécifié dans l'entête de celui-ci, toute référence externe d'un module du tableau de module étant interprétée comme une référence à l'index du module API.

7. Procédé de chargement d'applications sur un système embarqué selon la revendication 6, caractérisé en ce que la liaison statique d'un module est effectuée de telle sorte que la référence à une classe externe à un module non API dans le pseudocode intermédiaire est un index dans un tableau de l'entête du module, dont chaque entrée est un identificateur (MID) d'un module API référencé, le chargement dudit module sur la plate-forme cible comprenant le remplacement de ladite référence par le numéro de l'index du module API obtenu à partir de l'identificateur (MID) de la table d'association MID/IMi des modules API.

8. Système embarqué comprenant une machine virtuelle et une plate-forme API incluant des interfaces de programmation d'application, une mémoire non volatile fixe, une mémoire non volatile programmable ou modifiable, et une mémoire vive, caractérisé en ce que la mémoire non volatile programmable comprend au moins un module API comprenant des classes système et des modules de services, au moins un tableau de représentation des modules (TRM), dans lequel les modules sont indexés et une table (70, 104) associant l'index (IM) d'un module du tableau de représentation à l'identificateur (MID) dudit module, chaque module comprenant un tableau de représentation des classes (TRC), dans lequel les classes sont indexées et dans lequel chaque classe présente une référence à l'index (IM) de son module, chaque classe comprenant un tableau de représentation des attributs (TRA) et des méthodes (TRMe), dans lesquels les attributs et méthodes sont indexés, la référence à une méthode ou un attribut étant codée sur au moins trois multiplets (bytes) correspondant à une référence à une classe interne (II) ou externe (IE) au module, une référence

externe au module constituant l'index du module API dans le tableau de module, un numéro de classe (NCI) correspondant à l'index de la classe dans la table de représentation des classes du module, et un numéro de méthode (NM) ou d'attribut (NA) correspondant à l'index de la méthode ou de l'attribut dans le tableau de représentation des méthodes ou attributs de la classe du module.

9. Système embarqué selon la revendication 8, caractérisé en ce qu'il comporte des moyens de comparaison du premier multiplet des trois multiplets de codage de référence à une méthode ou à un attribut avec une valeur déterminée n pour décider s'il s'agit d'une classe interne ou externe.

10. Système embarqué selon la revendication 8, caractérisé en ce qu'il comprend un module principal comprenant le programme principal du système.

11. Système embarqué selon la revendication 8, caractérisé en ce que les classes sont indexées suivant l'ordre classes publiques puis classes en paquetages privés, et les attributs ou méthodes suivant l'ordre attribut ou méthode public, protégé, en paquetage privé et en classe privée.

12. Système embarqué selon la revendication 11, caractérisé en ce que la mémoire non volatile programmable comprend plusieurs modules API comprenant des classes système, deux tableaux de représentation (101, 103) des modules, l'un pour les modules API et l'autre pour les modules non-API, et le module principal, et deux tables (102, 104) d'association MID/IMi correspondant chacune à un tableau de représentation des modules.

13. Système embarqué selon la revendication 10, caractérisé en ce qu'il comprend une classe gestionnaire d'accès "Access manager" d'un module API (105) comprenant une méthode permettant de créer une instance d'un module de service, par l'intermédiaire du module principal, ladite classe présentant une protection lui interdisant d'avoir plus d'une instance.

14. Procédé d'exécution d'une application d'un système embarqué multi-application, comprenant un environnement d'exécution incluant une

machine virtuelle comprenant un interpréteur de langage de type pseudocode intermédiaire, et des interfaces de programmation d'application (API), caractérisé en ce que, lors de l'exécution du pseudocode intermédiaire d'un module de service, correspondant à une application, référencée dans un tableau de module, la référence à une méthode ou un attribut dans le pseudocode, codée sur au moins trois multiplets (bytes) correspondant à une référence à une classe interne (II) ou externe (IE) au module, un numéro de classe (NCI) et un numéro de méthode (NM) ou d'attribut (NA), une référence externe au module est interprétée par la machine virtuelle comme une référence à l'index d'un module API du tableau du ou des modules API .

15. Procédé d'exécution d'une application d'un système embarqué multi-application selon la revendication 14, caractérisé en ce que, sur réception d'une demande d'exécution d'un module de service présentant un identificateur (MID), l'environnement d'exécution accède à la classe d'entrée d'un module principal comprenant le programme principal du système, le module principal installe une instance d'une classe spéciale "Access Manager" d'un module API, gérant l'accès à un module de service, et utilise une méthode de cette classe permettant de créer une instance de la classe d'entrée du module de service demandée, par l'intermédiaire d'une table d'association de l'identificateur à l'index du module dans un tableau dans lequel le module est référencé, l'instance étant retournée par la méthode au programme principal.

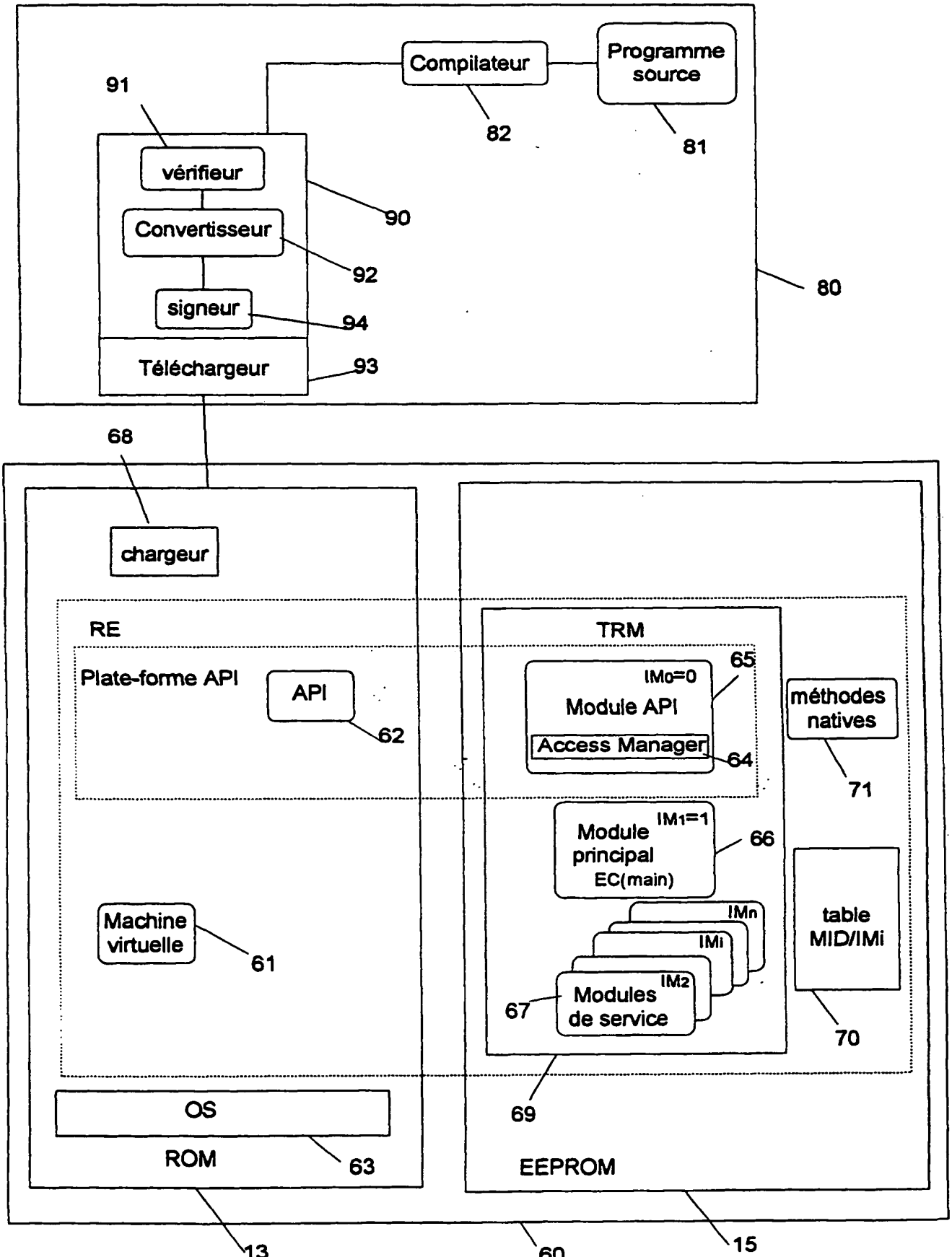

Compilateur
Programme source
91
82
81
vérifieur
Convertisseur
signeur
Téléchargeur
90
92
94
93
80
68
chargeur
RE
Plate-forme API
API
62
TRM
65
IMo=0
Module API
Access Manager
64
méthodes natives
71
IM1=1
Module principal
EC(main)
66
Machine virtuelle
61
IMn
IMi
IM2
Modules de service
67
table MID/IMi
70
69
OS
ROM
63
EEPROM
13
60
15

FIGURE 1

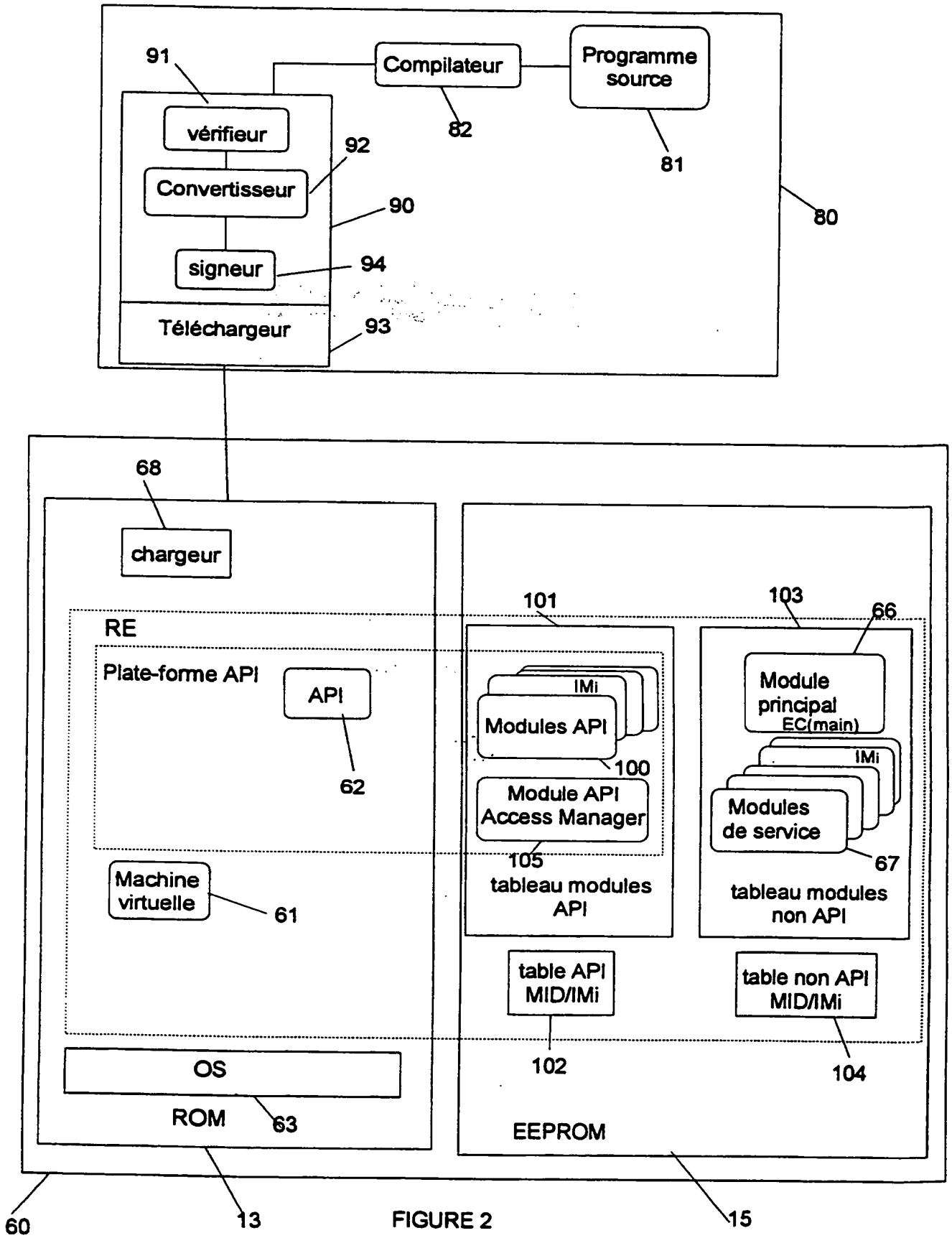

91
Compilateur
Programme source
vérifieur
92
82
81
Convertisseur
90
80
signeur
94
Téléchargeur
93
68
chargeur
101
103
66
RE
Plate-forme API
API
IMi
Modules API
100
Module API Access Manager
105
tableau modules API
Module principal EC(main)
IMi
Modules de service
67
tableau modules non API
62
Machine virtuelle
61
table API MID/IMi
table non API MID/IMi
102
104
OS
ROM
63
EEPROM
60
13
15

FIGURE 2

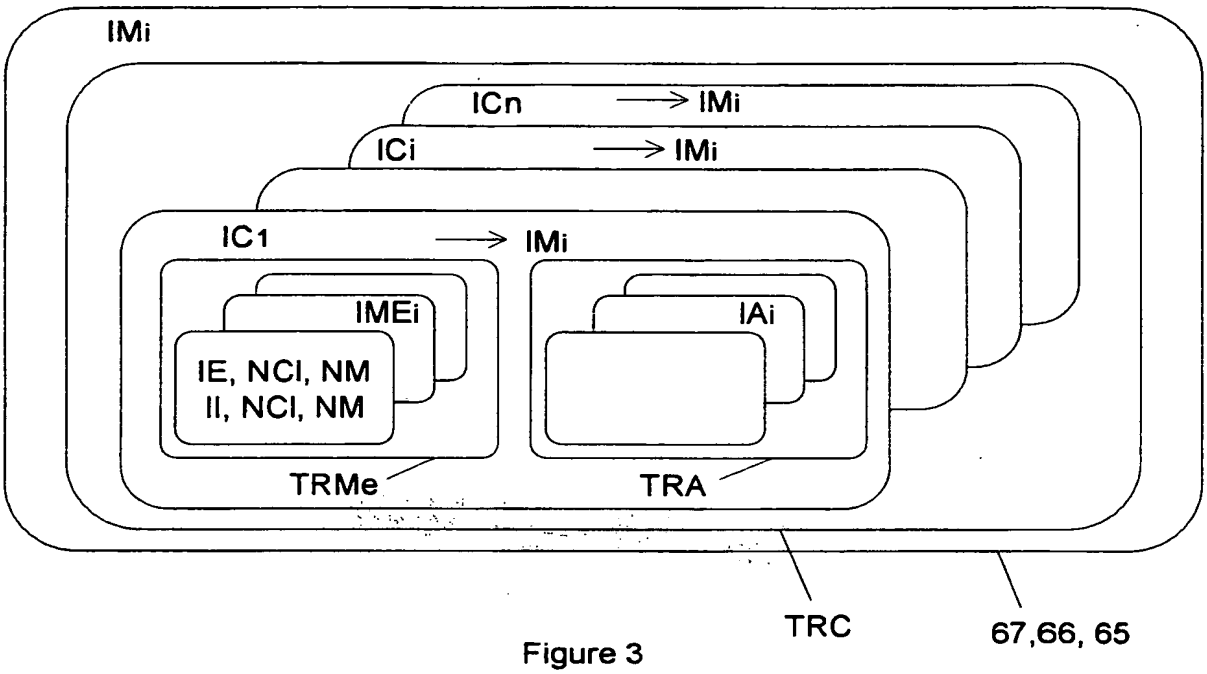


Figure 3

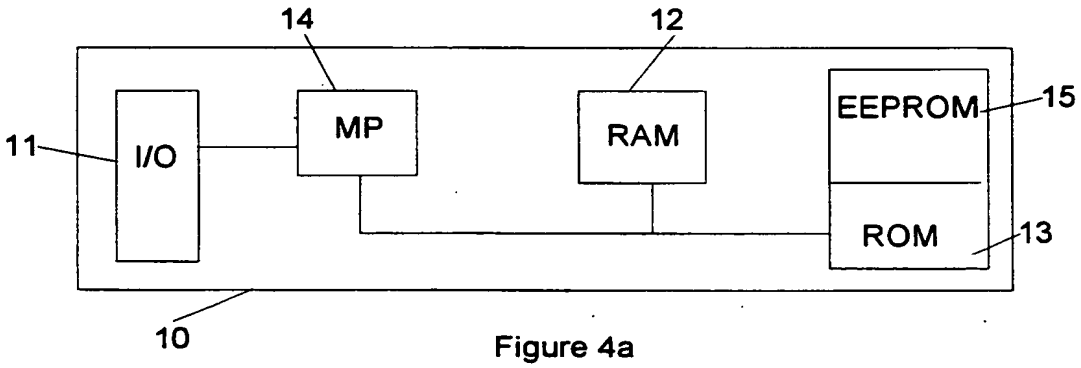


Figure 4a

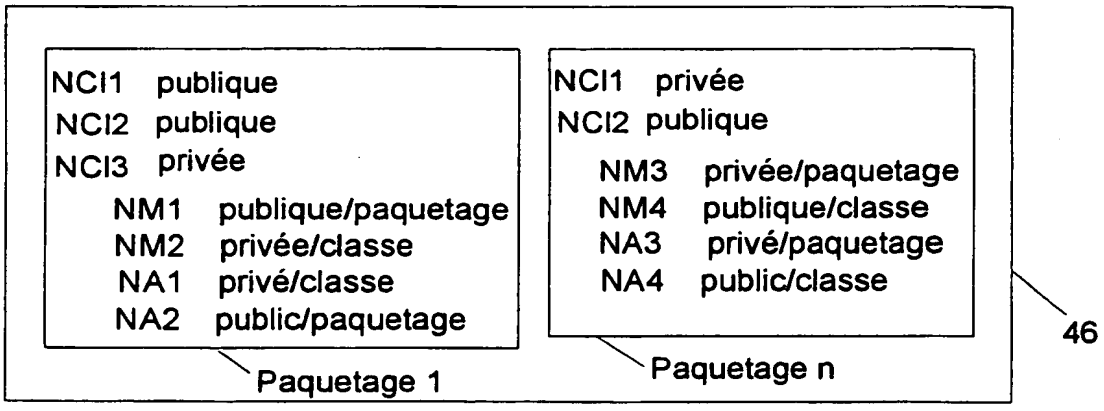


Figure 4c

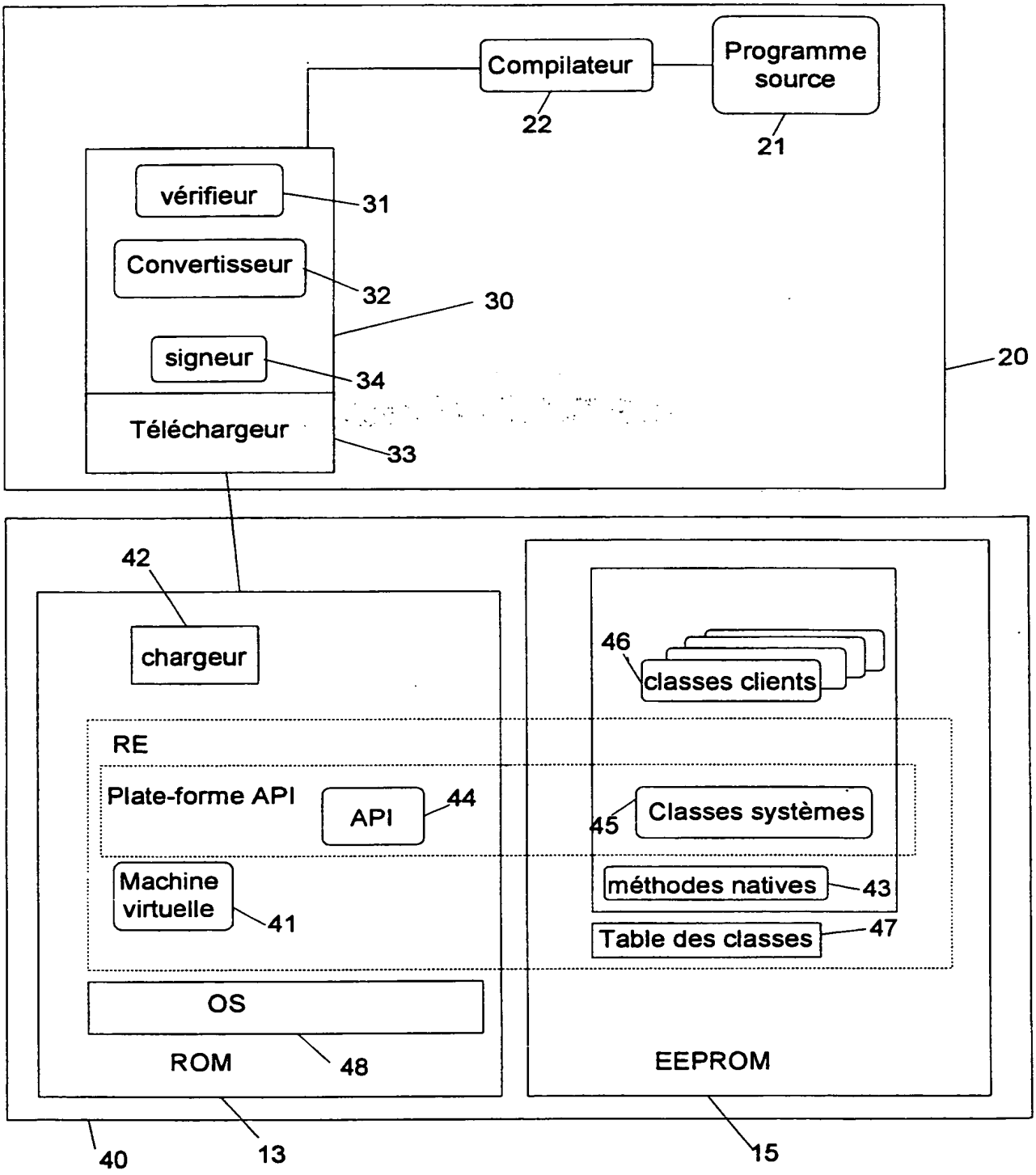

Compilateur
Programme source
22
21
vérifieur
31
Convertisseur
32
30
signeur
34
20
Téléchargeur
33
42
chargeur
46
classes clients
RE
Plate-forme API
API
44
45
Classes systèmes
Machine virtuelle
41
méthodes natives
43
Table des classes
47
OS
ROM
48
EEPROM
40
13
15

FIGURE 4b

INTERNATIONAL SEARCH REPORT

Interr ıal Application No
PCT/FR 00/03193

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/445 G07F7/10

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F G07F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, IBM-TDB, INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WO 98 19237 A (SCHLUMBERGER TECHNOLOGIES INC) 7 May 1998 (1998-05-07) page 6, line 30 -page 7, line 7 page 8, line 5 - line 19 page 9, line 32 -page 11, line 2 page 12, line 17 - line 29 page 16, line 5 - line 33 page 18, line 3 - line 28 page 31, line 26 -page 33, line 16 * Annexe A * ---	1-15
A	ZHIQUN CHEN, RINALDO DI GIORGIO: "Understanding Java Card 2.0" INTERNET DOCUMENT, March 1998 (1998-03), XP002146332 page 3, line 7 -page 5, line 43 --- -/--	1-15

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
22 January 2001	30/01/2001

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Ecolivet, S.

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Inter. nal Application No
PCT/FR 00/03193

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	FR 2 673 476 A (GEMPLUS CARD INT) 4 September 1992 (1992-09-04) page 1, line 25 -page 2, line 25	1-15

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inter. nal Application No
PCT/FR 00/03193

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9819237 A	07-05-1998	AU 722463 B	03-08-2000
		AU 4911897 A	22-05-1998
		EP 0932865 A	04-08-1999
		JP 2000514584 T	31-10-2000
FR 2673476 A	04-09-1992	DE 69205425 D	16-11-1995
		DE 69205425 T	21-03-1996
		EP 0589884 A	06-04-1994
		ES 2082451 T	16-03-1996
		WO 9213322 A	06-08-1992
		JP 6504862 T	02-06-1994
		US 5473690 A	05-12-1995

RAPPORT DE RECHERCHE INTERNATIONALE

Dem Internationale No
PCT/FR 00/03193

A. CLASSEMENT DE L'OBJET DE LA DEMANDE
CIB 7 G06F9/445 G07F7/10

Selon la classification internationale des brevets (CIB) ou à la fois selon la classification nationale et la CIB

B. DOMAINES SUR LESQUELS LA RECHERCHE A PORTE

Documentation minimale consultée (système de classification suivi des symboles de classement)
CIB 7 G06F G07F

Documentation consultée autre que la documentation minimale dans la mesure où ces documents relèvent des domaines sur lesquels a porté la recherche

Base de données électronique consultée au cours de la recherche internationale (nom de la base de données, et si réalisable, termes de recherche utilisés)
EPO-Internal, IBM-TDB, INSPEC

C. DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie °	Identification des documents cités, avec, le cas échéant, l'indication des passages pertinents	no. des revendications visées
A	WO 98 19237 A (SCHLUMBERGER TECHNOLOGIES INC) 7 mai 1998 (1998-05-07) page 6, ligne 30 -page 7, ligne 7 page 8, ligne 5 - ligne 19 page 9, ligne 32 -page 11, ligne 2 page 12, ligne 17 - ligne 29 page 16, ligne 5 - ligne 33 page 18, ligne 3 - ligne 28 page 31, ligne 26 -page 33, ligne 16 * Annexe A * ---	1-15
A	ZHIQUN CHEN, RINALDO DI GIORGIO: "Understanding Java Card 2.0" INTERNET DOCUMENT, mars 1998 (1998-03), XP002146332 page 3, ligne 7 -page 5, ligne 43 --- -/--	1-15

[X] Voir la suite du cadre C pour la fin de la liste des documents

[X] Les documents de familles de brevets sont indiqués en annexe

° Catégories spéciales de documents cités:

"A" document définissant l'état général de la technique, non considéré comme particulièrement pertinent

"E" document antérieur, mais publié à la date de dépôt international ou après cette date

"L" document pouvant jeter un doute sur une revendication de priorité ou cité pour déterminer la date de publication d'une autre citation ou pour une raison spéciale (telle qu'indiquée)

"O" document se référant à une divulgation orale, à un usage, à une exposition ou tous autres moyens

"P" document publié avant la date de dépôt international, mais postérieurement à la date de priorité revendiquée

"T" document ultérieur publié après la date de dépôt international ou la date de priorité et n'appartenenant pas à l'état de la technique pertinent, mais cité pour comprendre le principe ou la théorie constituant la base de l'invention

"X" document particulièrement pertinent; l'invention revendiquée ne peut être considérée comme nouvelle ou comme impliquant une activité inventive par rapport au document considéré isolément

"Y" document particulièrement pertinent; l'invention revendiquée ne peut être considérée comme impliquant une activité inventive lorsque le document est associé à un ou plusieurs autres documents de même nature, cette combinaison étant évidente pour une personne du métier

"&" document qui fait partie de la même famille de brevets

Date à laquelle la recherche internationale a été effectivement achevée	Date d'expédition du présent rapport de recherche internationale
22 janvier 2001	30/01/2001

Nom et adresse postale de l'administration chargée de la recherche internationale	Fonctionnaire autorisé
Office Européen des Brevets, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Ecolivet, S.

1

Formulaire PCT/ISA/210 (deuxième feuille) (juillet 1992)

RAPPORT DE RECHERCHE INTERNATIONALE

Dem: Internationale No
PCT/FR 00/03193

C.(suite) DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie °	Identification des documents cités, avec,le cas échéant, l'indicationdes passages pertinents	no. des revendications visées
A	FR 2 673 476 A (GEMPLUS CARD INT) 4 septembre 1992 (1992-09-04) page 1, ligne 25 -page 2, ligne 25	1-15

Formulaire PCT/ISA/210 (suite de la deuxième feuille) (juillet 1992)

RAPPORT DE RECHERCHE INTERNATIONALE

Renseignements relatifs aux membres de familles de brevets

Dem. Internationale No
PCT/FR 00/03193

Document brevet cité au rapport de recherche	Date de publication	Membre(s) de la famille de brevet(s)	Date de publication
WO 9819237 A	07-05-1998	AU 722463 B	03-08-2000
		AU 4911897 A	22-05-1998
		EP 0932865 A	04-08-1999
		JP 2000514584 T	31-10-2000
FR 2673476 A	04-09-1992	DE 69205425 D	16-11-1995
		DE 69205425 T	21-03-1996
		EP 0589884 A	06-04-1994
		ES 2082451 T	16-03-1996
		WO 9213322 A	06-08-1992
		JP 6504862 T	02-06-1994
		US 5473690 A	05-12-1995